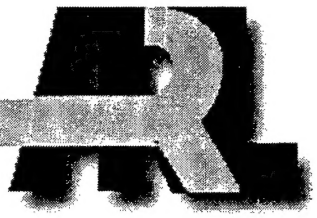
ARMY RESEARCH LABORATORY



A Class for Run Time Computation of Shadows for Polygonal Objects

Mark A. Thomas

ARL-TN-156 AUGUST 2000

Abstract

The development of realistic synthetic environments for dismounted soldier visualization requires attention to environmental cues. Effects such as shadows provide the soldier information about time of day and orientation. Graphical databases that do not have shadows must compute them at run time. Various methods of computing and displaying shadows exist. This report presents a simple method that is based on bounding volumes. The method provides the rapid generation of shadows. These shadows do not provide light attenuation effects, but the environmental cueing provides valuable information for the dismounted soldier in a synthetic environment.

TABLE OF CONTENTS

INTENTIONALLY LEFT BLANK

LIST OF FIGURES

INTENTIONALLY LEFT BLANK

LIST OF TABLES

INTENTIONALLY LEFT BLANK

A CLASS FOR RUN TIME COMPUTATION OF
SHADOWS FOR POLYGONAL OBJECTS

1. Introduction

Shadowing is a visualization technique that adds realism to computer-generated imagery. Shadows are an important part of the natural environment, providing information about sun position and altering the lighting characteristics of objects within the shadows. The lack of shadows in a scene reveals its synthetic nature and deprives the viewer of important visual clues.

This report describes a simple technique based on gross object characteristics. It has the advantage of being fast, producing a minimal number of polygons, and having a minimal impact on frame rate. The generated shadow polygon is used as a surface detail polygon, providing shading of the underlying objects and terrain without affecting lighting of objects within the shadow. The algorithm can be used during run time to revise shadows based on time-of-day changes.

This technique provides a visual context for the viewer. Simple shadows are used in today's video games and provide important realism for the player.

Real-time shadow visualization has a number of implications. For example, in combat simulation with semi-automated forces (SAF) and human-in-the-loop simulators, shadows must be considered when visibility and line of sight are computed. The human-in-the-loop simulation and the SAF must determine visibility in the same way or the simulation will not result in a "fair fight."

2. Computer-Generated Shadows

Computer-generated imaging can employ a variety of methods to compute shadows. Ray tracing produces the most realistic shadows but has the disadvantage of being computationally intensive. Ray tracing computes the correct shading and lighting parameters for all parts of a scene from one or multiple light sources. In simple ray tracing, a regular grid is created to determine the coordinates of the rays to be fired from the light source. Rays are recursively fired from the light source through the grid into the scene until all points in the grid have been used. The color of the displayed pixel is the color of the object with the closest intersection to the light source. When all rays have been evaluated, the scene is shaded.[1] Ray tracing requires intense computation and is impractical for real-time visualization on all but the most high powered

supercomputers and image generators. Research in this area is concentrating on improved parallelism, data throughput, and pixel drawing speed.

Shadow volumes require the development of a polygon representing the object's shadow, which is then used to determine whether the light source is occluded.[1] Unlike ray tracing, which evaluates every pixel of a scene, shadow volumes process polygonal edges. This technique requires the development of a shadow polygon, which is used to develop a shadow volume. The polygons of the shadow volume are tested relative to the light source. A front-facing polygon shades the object polygons behind it, and a back-facing shadow polygon means the object polygon is lit. Shadow volumes can be used in multi-pass algorithms, first to determine which polygons are lit, and then a finer computation of the actual shadow appearance. Shadow volumes require the development of the shadow volume, and the multi-pass nature of the algorithm requires further computation.

This report describes an implementation of a fast shadow projection technique, described in [2], which can be used for real-time polygonal database visualizations. The code is written in the C++ programming language. The class structure is graphics language dependent, based on Silicon Graphics, Inc. (SGI) Performer graphics programming language. The user must replace SGI Performer function calls with equivalent functionality for use in other graphics environments. The source code is included in the appendix.

3. Fake Shadows Technique

The "fake shadows" technique uses the bounding box of an object to compute a reasonable approximation of the shadow.[2] The shadow is a simple projection of the appropriate object edges on the ground. The technique computes the bounding box of the object, determines the projection on the ground, and creates a mesh of 12 polygons, which represents the shadow.

The algorithm uses the actual vertices of the object to create the shadow mesh. This makes the shadow appear to emanate from the object itself and not from an uncorrelated bounding box (see Figure 1). The shadow polygon is then placed on the shadowed object, anchored at the base by the appropriate vertices. The shadow is realistic for simple shapes (e.g., squares, parallelograms) and so includes a large number of objects in the natural environment such as buildings.

3.1 Bounding Box

The bounding box is determined using a "minimax" test. The object's bounding box is computed when the model is loaded into the visualization system via the SGI Performer library function call, *pfuTravCalcBBox()*. This bounding box is passed to the library as an argument, but computing the shadows from the gross

bounding box introduces artifacts. For example, the bounding box of the object is the minimum X, minimum Y, maximum X, and maximum Y. These values may or may not correspond to a vertex on the object itself but to components of many different vertices. Therefore, this library assumes that the actual vertices that correspond to this bounding box are unknown. This library attempts to match the actual object vertices with the gross bounding box so that the shadow appears to emanate from the object itself and not from the gross bounding box.

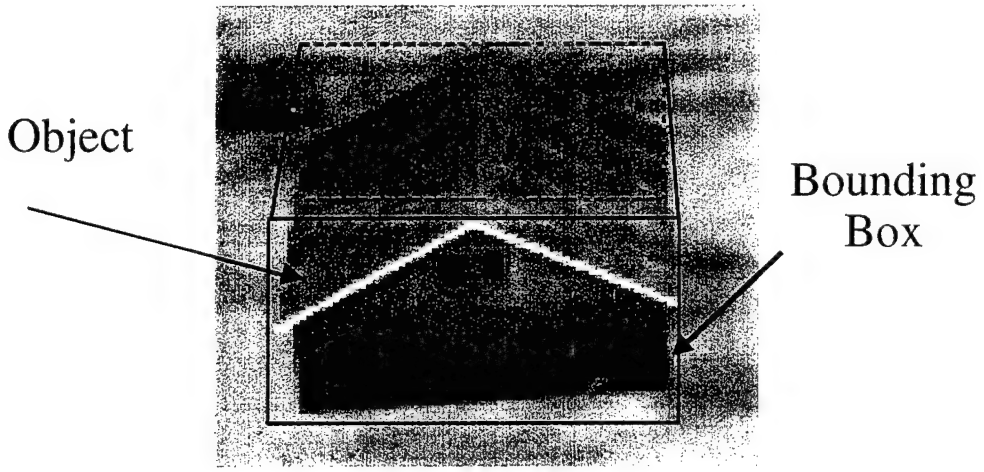


Figure 1. Bounding Box Example. (Note that the edges of the bounding box do not correspond with the actual profile of the building.)

The first step in determining the correct object vertices to use for the shadows is to orient the object on the ground with respect to the light source. The result is the Cartesian quadrant of the object with respect to the light source. Based on this quadrant, the bounding box vertices are combined to create five vertices. Table 1 shows how the five vertices are determined.

3.2 Ground Projection

When the five vertices are computed, these become the new bounding vertices. Vertices 1, 5, and 3 are used to compute the ground projection. The object vertices are tested against these three, and the closest vertices to them are used as anchor points for the shadow projection grid. The projection is a simple ray from the light source through the vertex to the ground. The three projection points are computed and stored.

3.3 Mesh Creation

The mesh is created from the three projection vertices in Paragraph 3.2 and from the bounding vertices 1 and 3 computed in Paragraph 3.1. The resulting five-vertex polygon represents the projection of the object onto the ground. However,

for certain objects, the sheer size of the projection presents problems. For example, sloping terrain presents a problem, as the ground can show through a large polygon when the terrain height changes within the bounds of the shadow polygon. Therefore, the five vertices are subdivided to create a finer mesh of 12 polygons, which more closely conforms to the ground. Figure 2 shows the resultant mesh with the vertex numbering.

Table 1

Determination of Five Vertices

Vertex	Quadrant 1	Quadrant 2	Quadrant 3	Quadrant 4
1	max X, max Y, min Z	min X, max Y, min Z	min X, min Y, min Z	max X, min Y, min Z
2	max X, max Y, max Z	min X, max Y, max Z	min X, min Y, max Z	max X, min Y, min Z
3	min X, max Y, max Z	min X, min Y, max Z	max X, min Y, max Z	max X, max Y, max Z
4	min X, min Y, max Z	max X, min Y, max Z	max X, max Y, max Z	min X, max Y, max Z
5	min X, min Y, min Z	max X, min Y, min Z	max X, max Y, min Z	min X, max Y, min Z

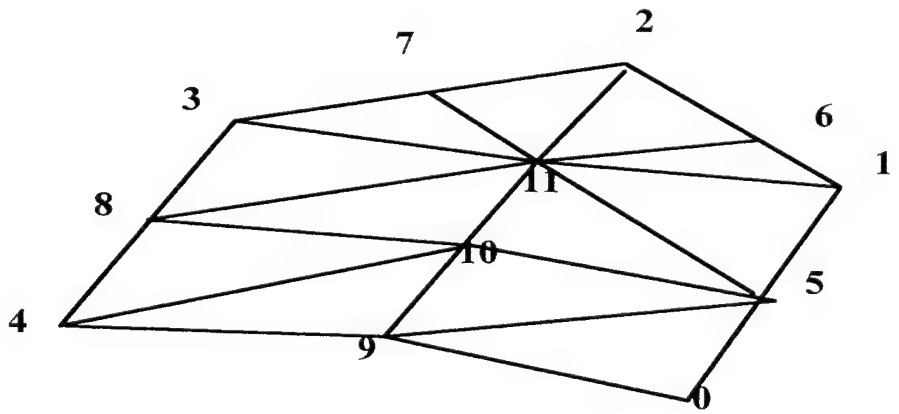


Figure 2. Shadow Vertex Numbering and Triangulation.

4. Graphics Display

The graphics engine used for this project is the SGI Performer graphics programming language.[3] The shadowing algorithm is graphics language independent, as long as the capability for computing the bounding box and extracting the object vertices is available.

The Performer libraries include a scene traversal mechanism to extract object-specific information from a loaded graphics object. This traverser allows the extraction of vertex coordinates, color, and texture values. The shadow library uses this capability to extract vertex coordinates and test them against the projection coordinates.

The shadow is enclosed in a Performer object called a GeoSet (geometry set). This GeoSet is attached to the scene graph for rendering.

The generated surface detail polygons have transparency, allowing the underlying geometry to show through, shaded by the shadow.

5. Sample Pictures

Figures 3, 4, and 5 show a scene with and without shadows.

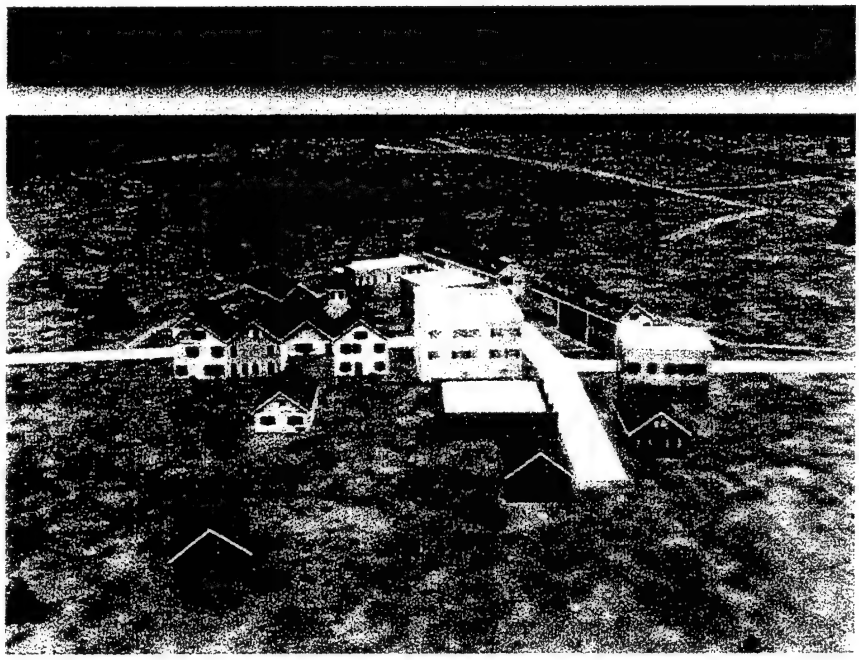


Figure 3. Terrain Database Visualization Without Shadows.

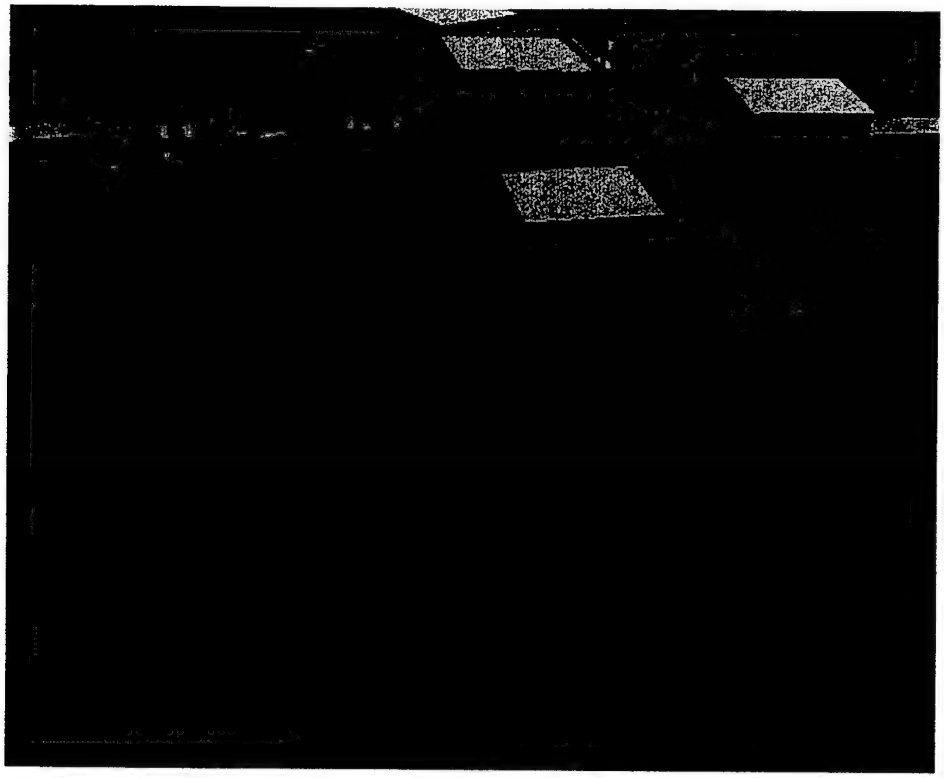


Figure 4. Terrain Database Visualization With Shadows Created by the Fake Shadows Technique.

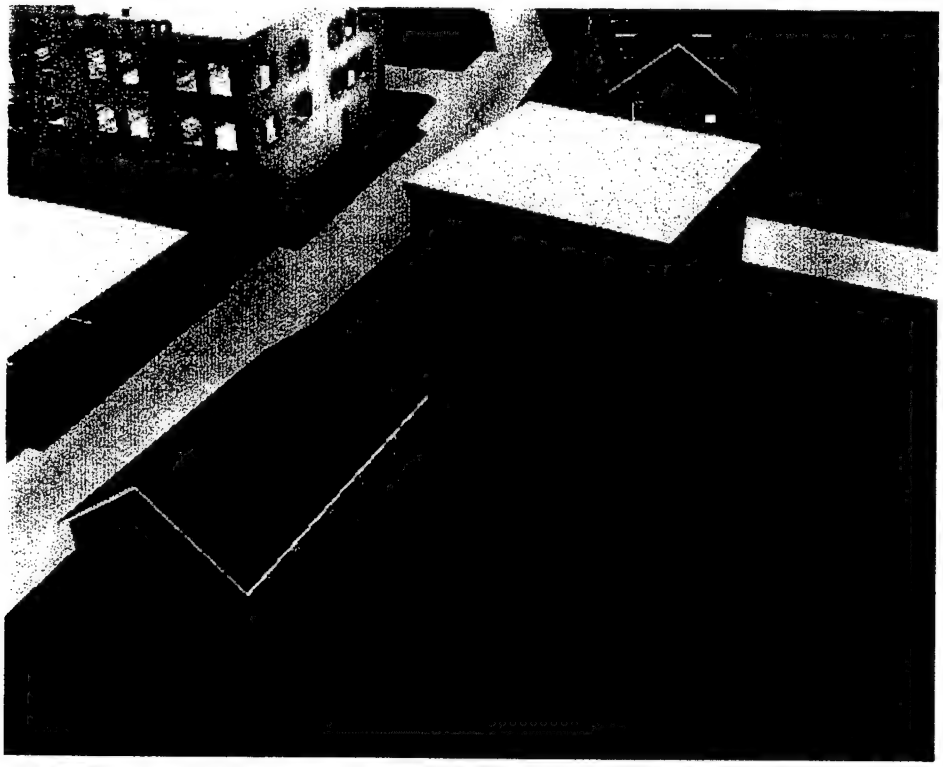


Figure 5. Close-up View of Buildings With Shadows. (Note how the terrain texture shows through the shadow.)

6. Conclusion

This algorithm provides for rapid, single-pass, polygonally based shadowing of arbitrary objects with a minimum of computation and additional polygons. It is easier to use and faster than ray-tracing methods and can be used in systems that do not provide shadowing through multi-pass rendering, such as PC base image generators. Limitations of this method include the inability to directly model the geometry of the shadowing surface, and the method does not modify the lighting of properties of polygons underneath the shadow. However, the shadow polygon can be used as a mask to shade polygons, given the appropriate programming methodology.

INTENTIONALLY LEFT BLANK

References

1. Foley, J., A. van Dam, S. Feiner, and J. Hughes, "Computer Graphics: Principles and Practice," Addison-Wesley Publishing Company, New York, 1993.

2. Adams, L., "High-Performance Graphics in C: Animation and Simulation," Wincrest Books, Blue Ridge Summit, PA, 1988.

3. Clay, Sharon et al., IRIS Performer 2.0 C++ reference pages, Document No. 007-2782-001, SGI, Mountain View, CA 1995.

INTENTIONALLY LEFT BLANK

APPENDIX A

SHADOW CLASS SOURCE CODE

INTENTIONALLY LEFT BLANK

SHADOW CLASS SOURCE CODE

This appendix contains the source code and header file for the shadow class. A sample calling sequence is given.

INTENTIONALLY LEFT BLANK

```
//*************************************************************************
// shadow.c - This file contains modules to compute shadows on objects
// The shadow computed is based on the bounding volume of the object.
// The shadow class definition is found in the shadow.h header
//*************************************************************************
#include <iostream.h>
#include <Performer/pf/pfGroup.h>

#include "shadow.h"

//
// void ComputeShadow() - This function computes a shadow for the
// given object.
//
void ComputeShadow(pfGroup *group, pfBox box, float ox, float oy,
    float oz, float heading, float lx, float ly, float lz )
{
    Shadow *shadow = new Shadow;

    heading *= 57.295f;

    shadow->setObjectPos( ox, oy, oz );
    shadow->setObjectOrientation( heading );
    shadow->setObjectBounds( box );
    shadow->setObjectGraphic( group );

// Compute the shadow
    shadow->compute( lx, ly, lz);
}
```

```
#include <Performer/pr/pfGeoMath.h>
#include <Performer/pr/pfGeoSet.h>
#include <Performer/pf/pfGeode.h>
#include <Performer/pr/pfMaterial.h>
#include <Performer/pfutil.h>
#include <math.h>

// Local Function Declarations
typedef struct _vdata{  // Vertex data for the traverser
    float x, y, z;      // The test point coordinates
    float rx, ry, rz;   // The returned vertex coordinatest
    float azimuth;      // Rotation of the object in world space
}VDATA;

extern float GetZ( float, float );
extern long DetectCollisionWithGround( float x, float y, float z,
    float h, float p, float len,float *rx, float *ry, float *rz,
    long mask );

class Shadow{
    public:
        Shadow( void );
        void print( void );
        void compute( float lx, float ly, float lz);
        void update( float sx, float sy, float sz );
        void setObjectPos( float x, float y, float z );
        void setObjectOrientation( float az );
        void setObjectBounds( pfBox box );
        void setObjectGraphic( pfGroup *group );
    private:
        void setVert( int i, float x, float y, float z );
        void setVertices( float sx, float sy, float sz );
        void computeAngle( float x1, float y1, float z1, float x2,
            float y2, float z2, float &h, float &p );
        void createGraphic( void );
        void getNearestVertex( float vx, float vy, float vz, float dx,
            float dy, float &x, float &y, float &z );
        pfGeode *getGraphic( void ){ return(this->geode); }
        void display( void );
        pfBox box;
        void CreateIList( ushort * );
        void CreateMesh( pfVec3 *verts, pfVec4 *colors );
        float pos[3];
        float azimuth;
        float vert[5][3];
        pfGroup *group;
        pfGeode *geode;
};
//
//
//
void Shadow::computeAngle( float x1, float y1, float z1,
    float x2, float y2, float z2, float &h, float &p)
{
    float dx, dy, dz, len;
    dy = y2 - y1;
    dx = x2 - x1;
    dz = z2 - z1;

    len = sqrt( dy * dy + dx * dx );
```

```
    h = fatan2( -dx, dy );

    if( h < 0.0f )
        h += 6.28;

    p = fatan2( dz, len );
}
//
//
//
void Shadow::compute( float lx, float ly, float lz )
{
    float x, y, z;
    float dx, dy;
    float ox = pos[0];
    float oy = pos[1];
    float oz = pos[2];
    float h, p;

    setVertices( lx, ly, lz );

    dx = ( box.max[0] - box.min[0] ) / 2.0;
    dy = ( box.max[1] - box.min[1] ) / 2.0;

// Get the nearest vertex on the object to the bounding box
// corner.  The shadow will be anchored to these points.
    getNearestVertex( vert[0][0] - ox, vert[0][1] - oy,
        GetZ( vert[0][0], vert[0][1] ), dx, dy, x, y, z );

    setVert( 0, x + ox, y + oy, GetZ( x + ox, y + oy ));

    vert[1][0] = x + ox; vert[1][1] = y + oy; vert[1][2] = box.max[2];

    getNearestVertex( vert[4][0] - ox, vert[4][1] - oy,
        GetZ( vert[4][0], vert[4][1] ), dx, dy, x, y, z );

    setVert( 4, x + ox, y + oy, GetZ( x + ox, y + oy ));
    vert[3][0] = x + ox; vert[3][1] = y + oy; vert[3][2] = box.max[2];

    getNearestVertex( vert[2][0] - ox, vert[2][1] - oy,
        GetZ( vert[2][0], vert[2][1] ), dx, dy, x, y, z );

    vert[2][0] = x + ox; vert[2][1] = y + oy; vert[2][2] = box.max[2];

// Compute the projection on the ground from the three vertices
    for( int i = 1; i < 4; i++ ){
        computeAngle(lx, ly, lz, vert[i][0],
            vert[i][1], vert[i][2], h, p );

        if( DetectCollisionWithGround( vert[i][0], vert[i][1],
            vert[i][2], h, p, 500.0, &x, &y, &z, 0 ) > 0 ){

            setVert( i, x, y, GetZ( x, y ) + 0.05f );
        }
        else{
            cout << "No Intersection Found For Vertex " << i << endl;
            cout << "No Shadow Made" << endl;
            return;
        }
```

```
    }

// Create the graphic object
    createGraphic();

// Add To The Scene
    display();
}
//
// Shadow Class Constructor
//
Shadow::Shadow( void )
{
    for( int i = 0; i < 5; i++ )
        this->setVert( i, -1.0, -1.0, -1.0 );

    geode = NULL;
    group = NULL;
}
//
//
//
void Shadow::setObjectGraphic( pfGroup *group )
{
    this->group = (pfGroup *)group->clone( 0 );
}
//
// Store the parent object's data
//
void Shadow::setObjectPos( float x, float y, float z )
{
    this->pos[0] = x;
    this->pos[1] = y;
    this->pos[2] = z;
}
//
//
//
void Shadow::setObjectOrientation( float az )
{
    this->azimuth = az;
}
//
//
//
void Shadow::setObjectBounds( pfBox inbox )
{
    for( int i = 0; i < 3; i++ ){
        box.min[i] = inbox.min[i];
        box.max[i] = inbox.max[i];
    }
}
//
// The shadow display function
//
void Shadow::display( void )
{
    extern pfGroup *root;
    root->addChild( (pfNode *)this->getGraphic() );
}
```

```
//
//
void Shadow::CreateIList( ushort *ilist )
{
    static ushort l[] = {
        0, 5, 9,
        5, 10, 9,
        10, 4, 9,
        8, 4, 10,
        5, 1, 10,
        10, 1, 11,
        11, 8, 10,
        11, 3, 8,
        1, 6, 11,
        6, 2, 11,
        2, 7, 11,
        7, 3, 11
    };

    for( int i = 0; i < 36; i++ )
        ilist[i] = l[i];

    return;
}
//
//
//
void Shadow::CreateMesh( pfVec3 *verts, pfVec4 *colors )
{
    extern float GetZ( float, float );
    pfVec3 tvec;

    for( int i = 0; i < 5; i++ ){
        tvec.sub( verts[i + 1], verts[i] );
        tvec.scale( 0.5, tvec );
        verts[i + 5].add( verts[i], tvec );
        verts[i + 5][2] = GetZ( verts[i + 5][0], verts[i + 5][1] ) + 0.1;
    }

    tvec.sub( verts[8], verts[5] );
    tvec.scale( 0.5, tvec );
    verts[10].add( verts[5], tvec );
    verts[10][2] = GetZ( verts[10][0], verts[10][1] ) + 0.1;

    tvec.sub( verts[3], verts[1] );
    tvec.scale( 0.5, tvec );
    verts[11].add( verts[1], tvec );
    verts[11][2] = GetZ( verts[11][0], verts[11][1] ) + 0.1;

    colors[5].copy( colors[0] );
    colors[6].copy( colors[1] );
    colors[7].copy( colors[3] );
    colors[8].copy( colors[4] );
    colors[9].copy( colors[0] );
    colors[10].copy( colors[0] );
    colors[11].copy( colors[1] );
}

#define NVERTS  12
```

```
void Shadow::createGraphic( void )
{
    pfGeoSet *gset;
    pfGeoState *gstate;
    pfMaterial *material;
    pfVec3 *verts;
    pfVec3 *norms;
    pfVec4 *colors;
    ushort *ilist;

    verts  = (pfVec3 *)pfMalloc( 3 * NVERTS * sizeof( pfVec3 ), pfGetSharedArena(
    norms  = (pfVec3 *)pfMalloc( 3 * NVERTS * sizeof( pfVec3 ), pfGetSharedArena(
    colors = (pfVec4 *)pfMalloc( 3 * NVERTS * sizeof( pfVec4 ), pfGetSharedArena(
    ilist  = (ushort *)pfMalloc( 3 * NVERTS * sizeof( ushort ), pfGetSharedArena(

    for( int i = 0; i < NVERTS; i++ ){
        norms[i].set( 0.0f, 0.0f, 1.0f );
    }

    verts[0].set( this->vert[0][0], this->vert[0][1], this->vert[0][2] );
    verts[4].set( this->vert[4][0], this->vert[4][1], this->vert[4][2] );
    verts[1].set( this->vert[1][0], this->vert[1][1], this->vert[1][2] );
    verts[2].set( this->vert[2][0], this->vert[2][1], this->vert[2][2] );
    verts[3].set( this->vert[3][0], this->vert[3][1], this->vert[3][2] );

    colors[0].set( 0.0, 0.0, 0.0, 0.5f );
    colors[4].set( 0.0, 0.0, 0.0, 0.5f );
    colors[1].set( 0.0, 0.0, 0.0, 0.5f );
    colors[2].set( 0.0, 0.0, 0.0, 0.5f );
    colors[3].set( 0.0, 0.0, 0.0, 0.5f );

    CreateMesh( verts, colors );
    CreateIList( ilist );

    gset = new pfGeoSet();

    gset->setNumPrims( NVERTS );
    gset->setPrimType( PFGS_TRIS );

    gset->setAttr( PFGS_COORD3,  PFGS_PER_VERTEX, verts,  ilist);
    gset->setAttr( PFGS_NORMAL3, PFGS_PER_VERTEX, norms,  ilist);
    gset->setAttr( PFGS_COLOR4,  PFGS_PER_VERTEX, colors, ilist);

    material = new pfMaterial();
    material->setColor( PFMTL_AMBIENT, 0.0f, 0.0f, 0.0f );
    material->setColor( PFMTL_DIFFUSE, 0.0f, 0.0f, 0.0f );
    material->setColorMode( PFMTL_FRONT, PFMTL_CMODE_AMBIENT_AND_DIFFUSE );
    material->setAlpha( 0.15f );

    gstate = new pfGeoState();
    gstate->setMode( PFSTATE_TRANSPARENCY, PFTR_ON );
    gstate->setAttr( PFSTATE_FRONTMTL, material );
    gstate->setAttr( PFSTATE_BACKMTL,  material );
    gstate->setMode( PFSTATE_ENTEXTURE, PF_OFF );
    gstate->setMode( PFSTATE_ENLIGHTING, PF_ON );

    gset->setGState( gstate );
    this->geode = new pfGeode();
    this->geode->addGSet( gset );
}
```

```
//
// Shadow Print Function
//
void Shadow::print( void )
{
    for( int i = 0; i < 5; i++ )
        cout << "Shadow Vertex " << i << " Is [" << this->vert[i][0] <<
            "," << this->vert[i][1] << "," << this->vert[i][2] << "]" << endl;
}
//
// Shadow Set Vertex Function
//
void Shadow::setVert( int i, float x, float y, float z )
{
    this->vert[i][0] = x;
    this->vert[i][1] = y;
    this->vert[i][2] = z;
}
//
//
//
void Shadow::setVertices( float sx, float sy, float sz)
{
    float h, p;
    int quad;

    computeAngle( sx, sy, sz, pos[0], pos[1], pos[2], h, p );

    quad = ( h * 57.295 ) / 90 + 1;

//    cout << "Heading and Pitch [" << h * 57.295 << "," <<
//      p * 57.295 << "]" << endl;


    switch( quad ){
        case 4:
            vert[0][0] = this->box.max[0]; vert[0][1] = this->box.min[1];
                vert[0][2] = this->box.min[2];
            vert[1][0] = this->box.max[0]; vert[1][1] = this->box.min[1];
                vert[1][2] = this->box.max[2];
            vert[2][0] = this->box.max[0]; vert[2][1] = this->box.max[1];
                vert[2][2] = this->box.max[2];
            vert[3][0] = this->box.min[0]; vert[3][1] = this->box.max[1];
                vert[3][2] = this->box.max[2];
            vert[4][0] = this->box.min[0]; vert[4][1] = this->box.max[1];
                vert[4][2] = this->box.min[2];
            break;
        case 1:
            vert[0][0] = this->box.max[0]; vert[0][1] = this->box.max[1];
                vert[0][2] = this->box.min[2];
            vert[1][0] = this->box.max[0]; vert[1][1] = this->box.max[1];
                vert[1][2] = this->box.max[2];
            vert[2][0] = this->box.min[0]; vert[2][1] = this->box.max[1];
                vert[2][2] = this->box.max[2];
            vert[3][0] = this->box.min[0]; vert[3][1] = this->box.min[1];
                vert[3][2] = this->box.max[2];
            vert[4][0] = this->box.min[0]; vert[4][1] = this->box.min[1];
                vert[4][2] = this->box.min[2];
            break;
        case 2:
```

```
            vert[0][0] = this->box.min[0]; vert[0][1] = this->box.max[1];
                vert[0][2] = this->box.min[2];
            vert[1][0] = this->box.min[0]; vert[1][1] = this->box.max[1];
                vert[1][2] = this->box.max[2];
            vert[2][0] = this->box.min[0]; vert[2][1] = this->box.min[1];
                vert[2][2] = this->box.max[2];
            vert[3][0] = this->box.max[0]; vert[3][1] = this->box.min[1];
                vert[3][2] = this->box.max[2];
            vert[4][0] = this->box.max[0]; vert[4][1] = this->box.min[1];
                vert[4][2] = this->box.min[2];
            break;
        case 3:
            vert[0][0] = this->box.min[0]; vert[0][1] = this->box.min[1];
                vert[0][2] = this->box.min[2];
            vert[1][0] = this->box.min[0]; vert[1][1] = this->box.min[1];
                vert[1][2] = this->box.max[2];
            vert[2][0] = this->box.max[0]; vert[2][1] = this->box.min[1];
                vert[2][2] = this->box.max[2];
            vert[3][0] = this->box.max[0]; vert[3][1] = this->box.max[1];
                vert[3][2] = this->box.max[2];
            vert[4][0] = this->box.max[0]; vert[4][1] = this->box.max[1];
                vert[4][2] = this->box.min[2];
            break;
    }
}
static int GetVertex( pfuTraverser *trav );
/*****************************************************************/
/* GetNearestVertex() - These next two functions traverse an     */
/* object's geometry and returns the vertex which is closest     */
/* to a given point.  The object's vertices are transformed      */
/* by the rotation of the object before the comparison.  The     */
/* rotated coordinate is passed back to the caller.              */
/*****************************************************************/
void Shadow::getNearestVertex( float x, float y, float z,
    float dx, float dy, float &rx, float &ry, float &rz )
{
    pfuTraverser trav;
    VDATA *vdata = (VDATA *)pfMalloc( sizeof( VDATA ), pfGetSharedArena() );
    pfuInitTraverser( &trav );

    vdata->x = x; vdata->rx = vdata->x + dx;
    vdata->y = y; vdata->ry = vdata->y + dy;
    vdata->z = vdata->rz = z;
    vdata->azimuth = this->azimuth;

    trav.preFunc = GetVertex;
    trav.data = (void *)vdata;

    pfuTraverse( this->group, &trav );

//    cout << "Nearest Vertex To " << x << "," << y << " Is " << vdata->rx << ","
//        << vdata->ry << endl;

    rx = vdata->rx; ry = vdata->ry;

    pfFree( vdata );
}
/*****************************************************************/
/* GetVertex() - The traverser callback routine.  This function*/
/* gets each vertex in the object and does the distance test   */
```

```
/*****************************************************************/
static int GetVertex( pfuTraverser *trav )
{
    VDATA *vdata = (VDATA *)trav->data;
    pfNode *node = (pfNode *)trav->node;
    pfMatrix mat;

// Make The Rotation Matrix
    mat.makeIdent();
    mat.makeRot( vdata->azimuth, 0.0, 0.0, 1.0f );

    if( node->isOfType( pfGeode::getClassType())){
        pfGeode *geode = (pfGeode *)node;
        int nGSets = geode->getNumGSets();
        float tx, ty, dist1;
        float dx, dy;
        tx = vdata->rx;
        ty = vdata->ry;

        dx = vdata->x - tx; // Deltas To The Test Point
        dy = vdata->y - ty;
        dist1 = dx * dx + dy * dy;

        for( int i = 0; i < nGSets; i++ ){
            int min, max;
            int start, stop;
            pfGeoSet *gset = geode->getGSet( i );
            int numPrims = gset->getNumPrims();
            int *primLens = gset->getPrimLengths();
            int ptype = gset->getPrimType();
            int nVerts = gset->getAttrRange( PFGS_COORD3, &min, &max );
            pfVec3 *verts;
            ushort *vindex;
            pfVec3 v;

            gset->getAttrLists( PFGS_COORD3, (void **)&verts, &vindex );

            if( min == 0 && max == -1 ){
                start = 0; stop = nVerts;
            }
            else{
                start = min; stop = max;
            }

// Find The Vertex Nearest The Test Vertex
            for( int j = start; j < stop; j++ ){
                float dx2, dy2;
                float dist2;

                v.copy( verts[j] );
                v.xformVec( v, mat );

                // If the signs don't match, they are not in the same quadrant.
                // Eliminate
                if( (vdata->x / v[0] ) < 0.0f || ( vdata->y / v[1] < 0.0f ))
                    continue;

                dx2 = v[0] - vdata->x;
                dy2 = v[1] - vdata->y;
```

```
                dist2 = dx2 * dx2 + dy2 * dy2;

                if( dist2 < dist1 ){
                    tx = v[0];
                    ty = v[1];
                    dist1 = dist2;
                }
            }
            vdata->rx = tx;
            vdata->ry = ty;
        }
    }

    return PFTRAV_CONT;
}
```

NO. OF COPIES ORGANIZATION

1 ADMINISTRATOR
DEFENSE TECHNICAL INFO CTR
ATTN DTIC OCA
8725 JOHN J KINGMAN RD
STE 0944
FT BELVOIR VA 22060-6218

1 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRL CI AI R REC MGMT
2800 POWDER MILL RD
ADELPHI MD 20783-1197

1 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRL CI LL TECH LIB
2800 POWDER MILL RD
ADELPHI MD 207830-1197

1 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRL D D SMITH
2800 POWDER MILL RD
ADELPHI MD 20783-1197

1 OFC OF THE SECY OF DEFNS
ATTN ODDRE (R&AT) G SINGLEY
THE PENTAGON
WASHINGTON DC 20301-3080

1 OSD
ATTN OUSD(A&T)/ODDDR&E(R)
ATTN R J TREW
THE PENTAGON
WASHINGTON DC 20310-0460

1 AMCOM MRDEC
ATTN AMSMI RD W C MCCORKLE
REDSTONE ARSENAL AL 35898-5240

1 CECOM
ATTN PM GPS COL S YOUNG
FT MONMOUTH NJ 07703

1 CECOM
SP & TERRESTRIAL COMMCTN DIV
ATTN AMSEL RD ST MC M
H SOICHER
FT MONMOUTH NJ 07703-5203

NO. OF COPIES ORGANIZATION

1 US ARMY INFO SYS ENGRG CMND
ATTN ASQB OTD F JENIA
FT HUACHUCA AZ 85613-5300

1 US ARMY NATICK RDEC
ACTING TECHNICAL DIR
ATTN SSCNC T P BRANDLER
NATICK MA 01760-5002

1 US ARMY RESEARCH OFC
4300 S MIAMI BLVD
RSCH TRIANGLE PARK NC 27709

1 US ARMY SIMULATION TRAIN & INSTRMNTN CMD
ATTN J STAHL
12350 RESEARCH PARKWAY
ORLANDO FL 32826-3726

1 US ARMY TANK-AUTOMOTIVE & ARMAMENTS CMD
ATTN AMSTA AR TD M FISETTE
BLDG 1
PICATINNY ARSENAL NJ
07806-5000

1 US ARMY TANK-AUTOMOTIVE CMD
RD&E CTR
ATTN AMSTA TA J CHAPIN
WARREN MI 48397-5000

1 US ARMY TRADOC
BATTLE LAB INTEGRATION & TECH DIR
ATTN ATCD B J A KLEVECZ
FT MONROE VA 23651-5850

1 NAV SURFACE WARFARE CTR
ATTN CODE B07 J PENNELLA
17320 DAHLGREN RD
BLDG 1470 RM 1101
DAHLGREN VA 22448-5100

1 DARPA
3701 N FAIRFAX DR
ARLINGTON VA 22203-1714

1 HICKS & ASSOCIATES, INC.
ATTN G SINGLEY III
1710 GOODRICH DR STE 1300
MCLEAN VA 22102

NO. OF
COPIES ORGANIZATION

1 HQ AFWA/DNX
106 PEACEKEEPER DR STE 2N3
OFFUTT AFB NE 68113-4039

1 US MILITARY ACADEMY
MATHEMATICAL SCIENCES CTR
OF EXCELLENCE
DEPT OF MATH SCIENCES
ATTN MDN A MAJ M D PHILLIPS
THAYER HALL
WEST POINT NY 10996-1786

2 US ARMY INST FOR THE
BEHAVIORAL SCIENCES
ATTN DR BRUCE KNERR
12350 RSCH PARKWAY
ORLANDO FL 32826-3276

2 US ARMY SIMULATION TRAIN &
INSTRMNTN CMD
ATTN J GROSSE P DUMANIOR
12350 RSCH PARKWAY
ORLANDO FL 32826-3276

ABERDEEN PROVING GROUND

2 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRL CI LP (TECH LIB)
BLDG 305 APG AA

10 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRL CI CD P JONES
M THOMAS A NEIDERER
G MOSS J FORESTER
R KASTE J OMAY
E HEILMAN C HANSEN
B BODT
BLDG 321

2 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRL HR SB P TRAN
P CROWELL
BLDG 518

NO. OF
COPIES ORGANIZATION

ABSTRACT ONLY

1 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRL CI AP TECH PUB BR
2800 POWDER MILL RD
ADELPHI MD 20783-1197

1 COMMANDER
US ARMY MATERIEL CMD
ATTN AMCRDA
5001 EISENHOWER AVENUE
ALEXANDRIA VA 22333-0001

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 2000	Final

4. TITLE AND SUBTITLE

A Class for Run Time Computation of Shadows for Polygonal Objects

5. FUNDING NUMBERS

PR: 9T1110

6. AUTHOR(S)

Thomas, M.A. (ARL)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
Computational and Information Sciences Directorate
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
Computational and Information Sciences Directorate
Aberdeen Proving Ground, MD 21005-5067

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ARL-TN-156

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The development of realistic synthetic environments for dismounted soldier visualization requires attention to environmental cues. Effects such as shadows provide the soldier information about time of day and orientation. Graphical databases that do not have shadows must compute them at run time. Various methods of computing and displaying shadows exist. This report presents a simple method that is based on bounding volumes. The method provides the rapid generation of shadows. These shadows do not provide light attenuation effects, but the environmental cueing provides valuable information for the dismounted soldier in a synthetic environment.

14. SUBJECT TERMS

computer graphics
simulation
terrain database
visualization

15. NUMBER OF PAGES
35

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	